MPI for Scalable Computing

Bill Gropp, University of Illinois at Urbana-Champaign

Rusty Lusk, Argonne National Laboratory

Rajeev Thakur, Argonne National Laboratory

The MPI Part of ATPESC

- We assume everyone has some MPI experience
- We will focus more on understanding MPI concepts than on coding details
- Emphasis will be on issues affecting scalability
- There will be some code walkthroughs and exercises
- We will use MPICH on your (Linux or MacOS) laptop for initial experiments
  - supports preliminary implementation of the new MPI-3 standard
- Vesta (BG/Q) will also be available for larger runs

Outline of MPI Material in ATPESC

- Today
  - MPI concepts
  - MPI-1, MPI-2, and MPI-3
  - Blocking and non-blocking communication
  - MPICH
  - Installing MPICH on your personal machine
  - Running some example code
- Tomorrow morning
  - Scalability issues in MPI programs
  - Sources of scalability problems
  - Avoiding communication delays
    - understanding synchronization
  - Minimizing data motion
    - using MPI datatypes
  - Topics in collective communication
- Tomorrow afternoon
  - Using remote memory access to avoid extra synchronization and data motion
  - The MPI-3 standard
  - The importance of process topologies
  - Example: neighborhood collectives
  - Work with halo exchange example

What is MPI?

- MPI is a message-passing library interface standard.
  - Specification, not implementation
  - Library, not a language
  - Classical message-passing programming model
- MPI-1 was defined (1994) by a broadly-based group of parallel computer vendors, computer scientists, and applications developers.
  - 2-year intensive process
- Implementations appeared quickly and now MPI is taken for granted as vendor-supported software on any parallel machine.
- Free, portable implementations exist for clusters and other environments (MPICH, Open MPI)

Timeline of the MPI Standard

- MPI-1 (1994), presented at SC'93
  - Basic point-to-point communication, collectives, datatypes, etc
- MPI-2 (1997)
  - Added parallel I/O, Remote Memory Access (one-sided operations), dynamic processes, thread support, C++ bindings, …
- ---- Unchanged for 10 years ----
- MPI-2.1 (2008)
  - Minor clarifications and bug fixes to MPI-2
- MPI-2.2 (2009)
  - Small updates and additions to MPI 2.1
- MPI-3 (2012)
  - Major new features and additions to MPI

Defining Some Terms

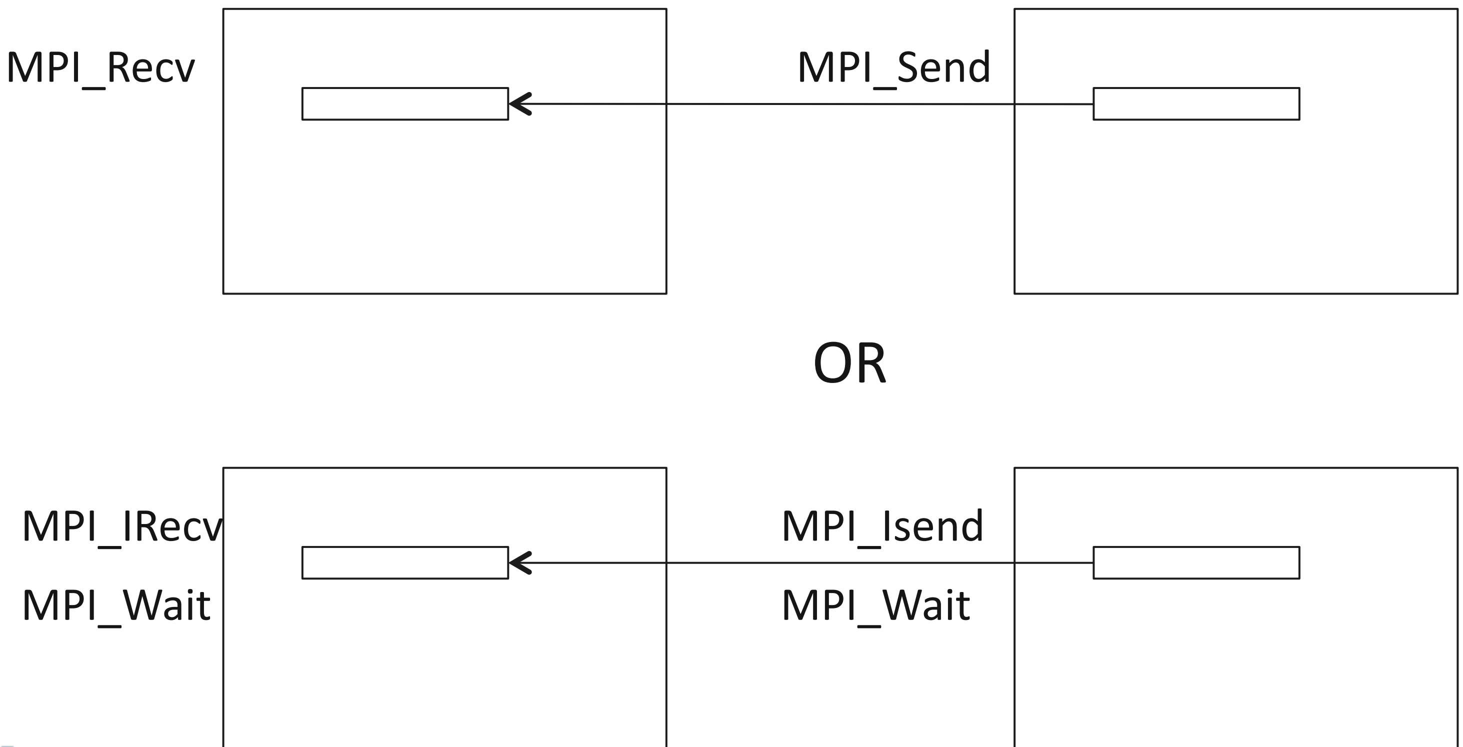
- A *process* consists of an address space, a program, and one or more threads of control, each with its own subroutine-call stack and program counter. The threads share the address space, which has advantages and disadvantages.
  - an old-fashioned Unix process is a single-threaded process.
- In MPI-1, a parallel program was thought of as a collection of old-fashioned Unix processes, each identified by its MPI *rank*.
  - Note that MPI was never SPMD (Single Program Multiple Data); different MPI ranks could always be executing different programs.
- In MPI-2, semantics were defined that enable MPI processes to be multithreaded (see "hybrid programming", later this week).

Programming and Address Spaces

- Sequential programming = one single-threaded process
- Parallel programming =
  - One process, multiple threads (OpenMP, pthreads) OR
  - Multiple single-threaded processes (MPI-1) OR
  - Multiple multiple-threaded processes (MPI-2)
- Shared-memory parallel programming is harder than it looks.
- Yet, processes (or threads) need to communicate, or else one has just a collection of sequential programs rather than a parallel program.
  - e.g., an old-fashioned batch system
- MPI is for communication among processes (with separate address spaces).

MPI Communication

- MPI limits in both time and space the exposure of one process's address space to action by (the threads of) another process



MPI Non-blocking Communication - 1

- MPI_Irecv exposes part of its address space to the "system" (OS + MPI implementation code + non-portable communication hardware/software)
  - the "system" may utilize internal buffers, perhaps smaller than the application's buffers, requiring multiple data transfers by the system
- MPI_Isend tells the system where the data is to be moved is located and into what process's receive buffer it is to be placed.
- Both buffers at this point belong to the "system".
- MPI_Wait on both sides delays its caller until the system no longer needs to access the buffer
  - Receiver can now make use of the new data in the buffer
  - Sender can now reuse the buffer

MPI Non-blocking Communication - 2

- The blocking operations (MPI_Send, MPI_Recv) can be dangerous.
  - The MPI Forum only included them because users of earlier systems would expect them.
- Deadlock danger: exchanging large messages

0	1
MPI_Send(1)	MPI_Send(0)
MPI_Recv(1)	MPI_Recv(0)

  - Deadlocks if the system cannot absorb the sent message, thus allowing the send to complete before the corresponding receive is posted.
- Performance danger: delayed receive

0	1
MPI_Send(1)	.
.	.
.	.
.	MPI_Recv(0)

  - Send blocks until corresponding receive is posted, perhaps much later.

Non-blocking Communication - 3

- Using the non-blocking receive (MPI_Irecv) solves both problems by providing the system a place on the receiving side to put the message when it is needed by the send.

0	1
MPI_Irecv(1)	MPI_Irecv(0)
MPI_Send(1)	MPI_Send(0)
MPI_Wait	MPI_Wait

and

0	1
MPI_Send(1)	MPI_Irecv(0)
	.
	.
	MPI_Wait

- Such a place can be provided on the sending side by the use of the buffered send (MPI_Bsend).

Overlapping Communication and Computation

- Some believe that the purpose of non-blocking communication is to specify that communication and computation are occur simultaneously, and are disappointed when it doesn't always happen.
- Non-blocking communication *allows* an implementation to do this if the "system" (hardware, MPI implementation, specialized communication software) can do so, but the real purpose is as described above.
- A standard-conforming MPI implementation on a specific platform is allowed to
  - Utilize a system thread or hardware support in order to move data in parallel with local computation between the Isend/Irecv and the Wait.
  - Move all or part of the message during some other MPI call (e.g., MPI_Test) between the Isend/Irecv and the Wait.
  - Complete an operation during the Isend call (if the "system" can absorb the message or the Irecv has been posted.
  - Delay the initiation of the data transfer until the corresponding Wait.

Summary of Types of Send

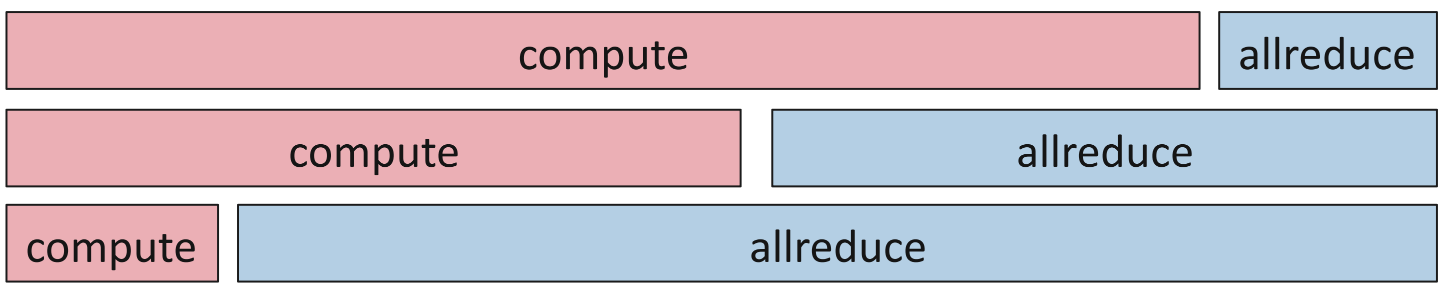
- MPI_Send blocks until the message has been absorbed by the "system". This does not mean that the message has been received.
- MPI_Isend doesn't block (should always return quickly).
- MPI_Ssend blocks until a matching receive has been posted (supplying the space for the message).
- MPI_Rsend *assumes* that the corresponding receive has been posted. The programmer is responsible.

0	1
MPI_Irecv(answer,1)	
MPI_Send(question,1)	MPI_Recv(question,0)
	MPI_Rsend(answer,1)
MPI_Wait	

- MPI_Bsend copies the message into a local buffer (provided by the user with MPI_Buffer_attach) in order to avoid blocking.

Collective Operations

- MPI provides many collective communication patterns, some with computation included. Custom computation operations are possible.
- Multiple algorithms based on messages sizes, machine topologies, machine capabilities.
  - Scalable algorithms a research topic
- Common feature: called by all processes in a communicator
- Performance note: Measuring time taken by a collective operation can obscure what is really a load balancing problem.



- MPI-3 has non-blocking collective operations.

MPI-2

- MPI-2 introduced dynamic process management, remote memory access (one-sided operations), parallel I/O, thread safety, C++ (since removed) and Fortran-90 bindings.
- We won't discuss here dynamic process management (not universally implemented, particularly on large systems, since it involves process management at the OS level).
- Thread safety will be covered under Hybrid Programming, later.
- A very brief conceptual discussion of RMA is here…

MPI-2 RMA: Remote Memory Access, or One-sided Operations

- The RMA *window object* can be thought of as a generalization of the MPI-1 communication buffer.
- Allocating a window object exposes a larger part of a process's address space for access by other processes, and (usually) for a longer time.
  – room for multiple, simultaneously active communication buffers.
  – MPI window = union of all process's window objects
- Separates "buffer" allocation, data movement initiation, and synchronization (checking for completion).

  MPI_Win_create

  MPI_Put
MPI_Get
MPI_Accumulate

  All are non-blocking; multiple operations can be active in same window object simultaneously

  MPI_Fence, Post-Start-Complete-Wait, Lock-Unlock

- More on RMA tomorrow…

MPI-2 Parallel I/O

- MPI_IO is based on an analogy: Reading from and writing to files is “like” receiving and sending messages from/to the (parallel) file system.
- Concepts from MPI-1 are reused:
  - datatypes to describe non-contiguous data (in memory and in files)
  - non-blocking operations
  - collective operations
- More on parallel I/O later this week
- MPI-3 tomorrow

End of General MPI Part

One Specific MPI Implementation -- MPICH

What is MPICH?

- MPICH is a high-performance and widely portable implementation of MPI
- It provides all features of MPI that have been defined so far (including MPI-1, MPI-2.0, MPI-2.1, MPI-2.2, and (almost all of) MPI-3.0)
- Serves as foundation for most vendor MPI implementations
- Active development lead by Argonne National Laboratory and University of Illinois at Urbana-Champaign
  - Several close collaborators who contribute many features, bug fixes, testing for quality assurance, etc.
    - IBM, Microsoft, Cray, Intel, Ohio State University, Queen's University, Myricom and many others
- Current release is MPICH-3.0.4
- Can run experiments here on your Linux or MacOS laptop or a cluster back home

Getting Started with MPICH

- Download MPICH
  - Go to http://www.mpich.org and follow the downloads link
  - or http://etherpad.mozilla.org/anl-training and follow link at bottom.
  - The download will be a zipped tarball
  - You don't have to download hydra as well, it is included in MPICH.
- Build MPICH
  - Unzip/untar the tarball:
  - `tar -xzvf mpich-3.0.4.tar.gz`
  - `cd mpich-3.0.4`
  - `./configure --prefix=/where/to/install/mpich |& tee c.log`
  - `make |& tee m.log`
  - `make install |& tee mi.log`
  - `Add /where/to/install/mpich/bin to your PATH`
- If there is no Fortran compiler on your machine, add

  `--disable-fc --disable-f77` to the configure line

Compiling MPI programs with MPICH

- Compilation Wrappers
  - For C programs: `mpicc mytest.c –o mytest`
  - For C++ programs: `mpicxx mytest.cpp –o mytest`
  - For Fortran 77 programs: `mpif77 mytest.f –o mytest`
  - For Fortran 90 programs: `mpif90 mytest.f90 –o mytest`
- You can link other libraries are required too
  - To link to a math library: `mpicc mytest.c –o mytest -lm`
- You can just assume that “mpicc” and friends have replaced your regular compilers (gcc, gfortran, etc.)

Running MPI programs with MPICH

- Launch 16 processes on the local node (e.g. your laptop):
  - `mpiexec -np 16 ./test`
- Launch 16 processes on 4 nodes (each has 4 cores)
  - `mpiexec -hosts h1:4,h2:4,h3:4,h4:4 -np 16 ./test`
    - Runs the first four processes on h1, the next four on h2, etc.
  - `mpiexec -hosts h1,h2,h3,h4 -np 16 ./test`
    - Runs the first process on h1, the second on h2, etc., and wraps around
    - So, h1 will have the 1st, 5th, 9th and 13th processes
- If there are many nodes, it might be easier to create a host file
  - `cat hf`

    ```
    h1:4
    h2:2
    ```
  - `mpiexec -hostfile hf -np 16 ./test`

Trying some example programs

- MPICH comes packaged with several example programs using almost ALL of MPICH's functionality
- A simple program to try out is the pi example written in C (cpi.c) – calculates the value of π in parallel (available in the examples directory when you build MPICH)
  - `mpiexec -np 16 ./examples/cpi`
- The output will show how many processes are running, and the error in calculating π
- Next, try it with multiple hosts
  - `mpiexec -hosts h1:2,h2:4 -np 16 ./examples/cpi`
- If things don't work as expected, send an email to discuss@mpich.org

Interaction with Resource Managers

- Resource managers such as SGE, PBS, SLURM or Loadleveler are common in many managed clusters
  - MPICH automatically detects them and interoperates with them
- For example with PBS, you can create a script such as:

```
#! /bin/bash

cd $PBS_O_WORKDIR
# No need to provide –np or –hostfile options
mpiexec ./test
```

- Job can be submitted as: `qsub –l nodes=2:ppn=2 test.sub`
  - "mpiexec" will automatically know that the system has PBS, and ask PBS for the number of cores allocated (4 in this case), and which nodes have been allocated
- The usage is similar for other resource managers

Running on BG/Q

scp cpi.c you@vesta.alcf.anl.gov:

See

http://www.alcf.anl.gov/user-guides/overview-how-compile-and-link

ssh vesta.alcf.anl.gov

Add +mpiwrapper-xl to ~/.soft file (if not already there)

Run the command "resoft"

mpixlc -o cpi cpi.c

See http://www.alcf.anl.gov/user-guides/how-queue-job

qsub -A ATPESC2013 -n 10 -t 10 ./cpi

Run qstat to see status in queue

Output will be in "job_number".output file

Debugging MPI programs

- Parallel debugging is trickier than debugging serial programs
  - Many processes computing; getting the state of one failed process is usually hard
  - MPICH provides in-built support for some debugging
  - And it natively interoperates with commercial parallel debuggers such as Totalview and DDT
- Using MPICH with totalview:
  - `totalview –a mpiexec –np 6 ./test`
- Using MPICH with ddd (or gdb) on one process:
  - `mpiexec –np 4 ./test : -np 1 ddd ./test : -np 1 ./test`
  - Launches the 5th process under “ddd” and all other processes normally

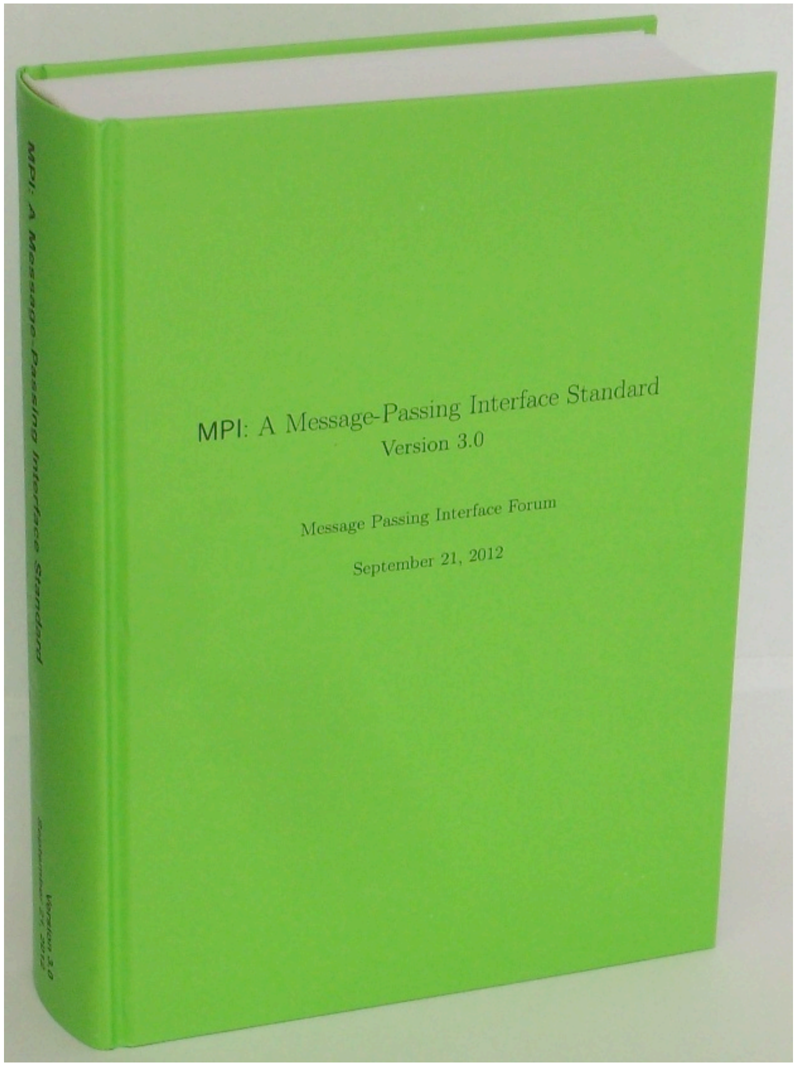
MPI Sources

- The Standard itself:
  - At http://www.mpi-forum.org
    - All MPI official releases. Latest version is MPI 3.0
    - Download pdf versions
- Online Resources
  - http://www.mcs.anl.gov/mpi
    - pointers to lots of stuff, including other talks and tutorials, a FAQ, other MPI pages
  - Tutorials: http://www.mcs.anl.gov/mpi/learning.html
  - Google search will give you many more leads

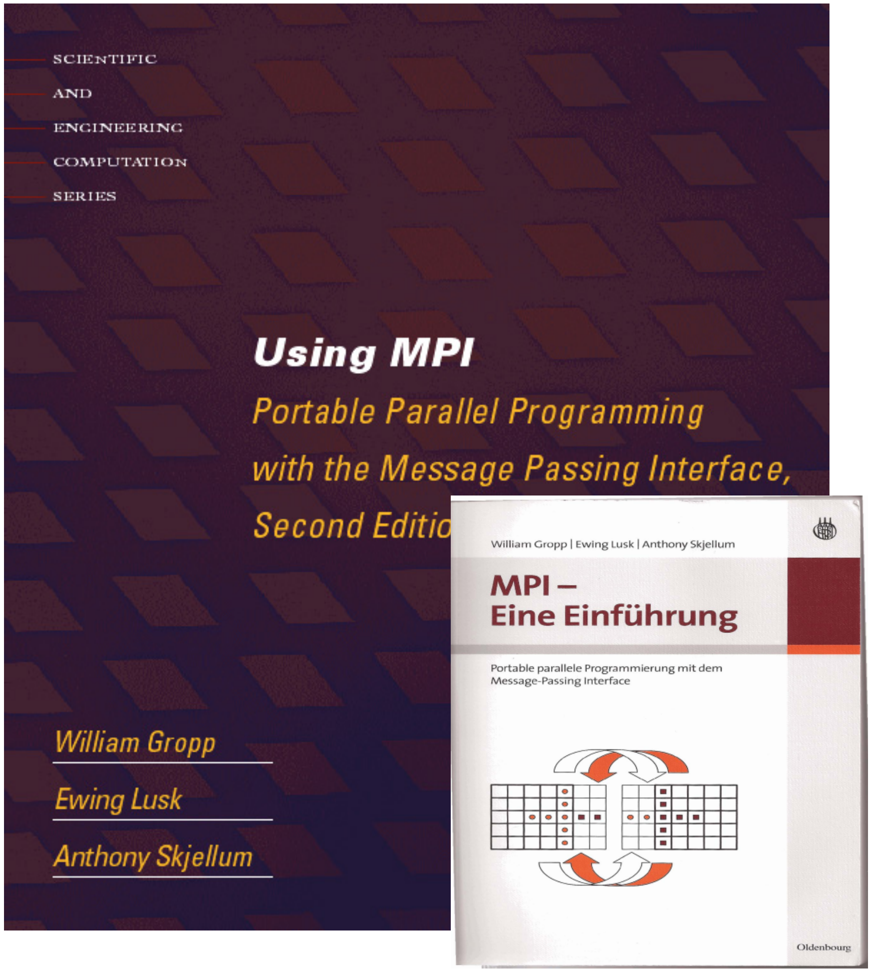
Latest MPI 3.0 Standard in Book Form

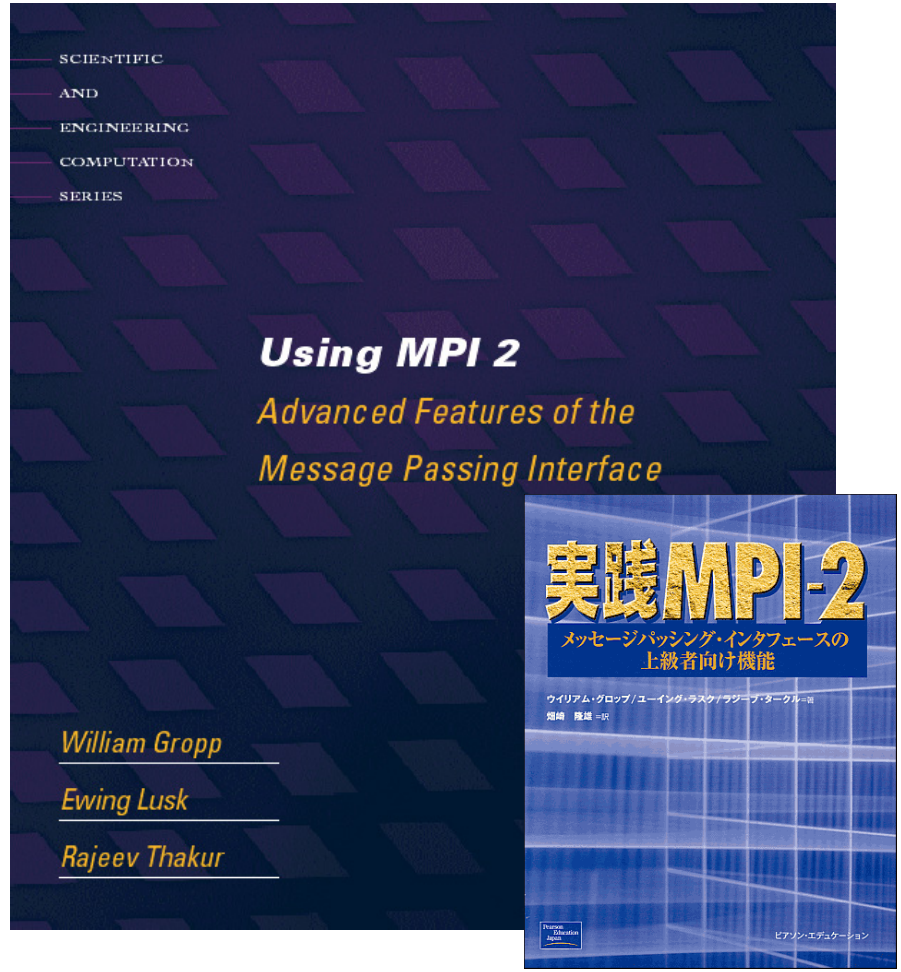
Available from amazon.com

http://www.amazon.com/dp/B002TM5BQK/



Tutorial Material on MPI, MPI-2





http://www.mcs.anl.gov/mpi/{usingmpi,usingmpi2}

Some Example Codes

www.cs.illinois.edu/~wgropp/advmpi.tgz

The End

MPI-3

Overview of New Features in MPI-3

- Major new features
  - Nonblocking collectives
  - Neighborhood collectives
  - Improved one-sided communication interface
  - Tools interface
  - Fortran 2008 bindings
- Other new features
  - Matching Probe and Recv for thread-safe probe and receive
  - Noncollective communicator creation function
  - "const" correct C bindings
  - Comm_split_type function
  - Nonblocking Comm_dup
  - Type_create_hindexed_block function
- C++ bindings removed
- Previously deprecated functions removed

Nonblocking Collectives

- Nonblocking versions of all collective communication functions have been added
  - MPI_Ibcast, MPI_Ireduce, MPI_Iallreduce, etc.
  - There is even a nonblocking barrier, MPI_Ibarrier
- They return an MPI_Request object, similar to nonblocking point-to-point operations
- The user must call MPI_Test/MPI_Wait or their variants to complete the operation
- Multiple nonblocking collectives may be outstanding, but they must be called in the same order on all processes

Neighborhood Collectives

- New functions MPI_Neighbor_allgather, MPI_Neighbor_alltoall, and their variants define collective operations among a process and its neighbors
- Neighbors are defined by an MPI Cartesian or graph virtual process topology that must be previously set
- These functions are useful, for example, in stencil computations that require nearest-neighbor exchanges
- They also represent sparse all-to-many communication concisely, which is essential when running on many thousands of processes.
  - Do not require passing long vector arguments as in MPI_Alltoallv

Improved Remote Memory Access Interface

- Substantial extensions to the MPI-2 RMA interface (MPI_Put, MPI_Get)
- New window creation routines:
  - MPI_Win_allocate: MPI allocates the memory associated with the window (instead of the user passing allocated memory)
  - MPI_Win_create_dynamic: Creates a window without memory attached. User can dynamically attach and detach memory to/from the window by calling MPI_Win_attach and MPI_Win_detach
  - MPI_Win_allocate_shared: Creates a window of shared memory (within a node) that can be can be accessed simultaneously by direct load/store accesses as well as RMA ops
- New atomic read-modify-write operations
  - MPI_Get_accumulate
  - MPI_Fetch_and_op (simplified version of Get_accumulate)
  - MPI_Compare_and_swap

Improved RMA Interface contd.

- A new “unified memory model” in addition to the existing memory model, which is now called “separate memory model”
- The user can query (via MPI_Win_get_attr) whether the implementation supports a unified memory model (e.g., on a cache-coherent system), and if so, the memory consistency semantics that the user must follow are greatly simplified.
- New versions of put, get, and accumulate that return an MPI_Request object (MPI_Rput, MPI_Rget, …)
- User can use any of the MPI_Test/Wait functions to check for local completion, without having to wait until the next RMA sync call

Tools Interface

- Beyond the PMPI profiling interface
- An extensive interface to allow tools (debuggers, performance analyzers, etc.) to portably extract information about MPI processes
- Enables the setting of various control variables within an MPI implementation, such as algorithmic cutoff parameters
  - e.g, eager v/s rendezvous thresholds
  - Switching between different algorithms for a collective communication operation
- Provides portable access to performance variables that can provide insight into internal performance information of the MPI implementation
  - e.g., length of unexpected message queue
- Note that each implementation defines its own performance and control variables; MPI does not define them

Fortran 2008 Bindings

- An additional set of bindings for the latest Fortran specification
- Supports full and better quality argument checking with individual handles
- Support for choice arguments, similar to (void *) in C
- Enables passing array subsections to nonblocking functions
- Optional ierr argument
- Fixes many other issues with the old Fortran 90 bindings

Miscellaneous Features

- Other new features
  - Matching Probe and Recv for thread-safe probe and receive
  - Noncollective communicator creation function
  - “const” correct C bindings
  - Comm_split_type function
  - Nonblocking Comm_dup
  - Type_create_hindexed_block function
- C++ bindings removed
- Previously deprecated functions removed

What did not make it into MPI-3

- Some evolving proposals did not make it into MPI-3
  - e.g., fault tolerance and improved support for hybrid programming
- This was because the Forum felt the proposals were not ready for inclusion in MPI-3
- These topics may be included in a future version of MPI

- Current activities of the MPI Forum (for MPI 3.x and MPI 4) can be tracked at http://meetings.mpi-forum.org/
- The full standard and other materials can be found at http://mpi-forum.org